An Embedded Wireless Mini-Server with Database Support

Hungchi Chang, Sy-Yen Kuo and Yennun Huang

Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan, R.O.C.

Abstract

Due to the maturity and popularity of wireless technologies, more and more digital devices support the wireless accessibility as an essential function. As a result, many emerging mobile services and applications are being implemented on such devices. In this paper, we describe an embedded wireless mini-server platform with database support which can be used to build many wireless applications and services. Our goal is to make the mini-server platform small in dimension, low in cost and power consumption, and very easy to develop new mobile applications on.

1 Introduction

The main goal of this project is to develop an easy-to-use and low cost embedded wireless and wired mini-server platform with networking and database support. Developers can use the platform to implement many integrated wireless services such as restaurant ordering systems. Users can access these services with a friendly user interface via a wired or wireless network. In addition, users can access Internet through this mini-server platform.

To provide a friendly user interface to users, a web server with CGI support is integrated into the system. Users can interact with the mini-server using web browsers. As a result, the services built by the mini-server platform are very easy to use and maintain.

In this paper, we will go over the system architecture, describe the cross-platform development environment, illustrate the use of the development environment, outline the main system components, and discuss the networking services provided by the platform. A restaurant ordering system using the platform will also be described.

2 System Architecture

The embedded hardware platform we use is FW-6410A with VIA Eden VE 400 MHz low-power processor. It has four Ethernet connectors, and each one has its own Realtek RTL8100 chip. It also supports a few standard interfaces, such as Compact Flash, RS-232, mini-PCI, and PCI. In the system, we use a 64 MB RAM and a 128 MB Compact Flash card as the main storage media, and an extra wireless network card that will function as an access point.

The architecture of the platform is given in figure 1.

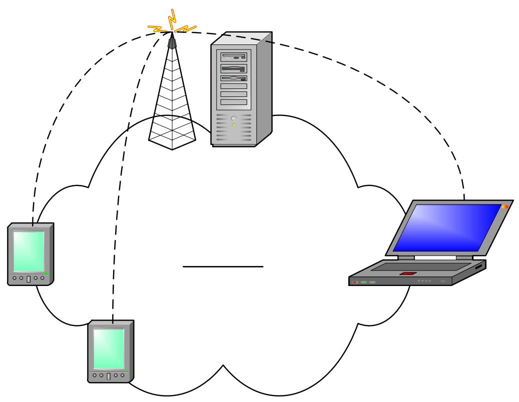


Figure 1.
The architecture of the embedded mini-server system

Figure 1 also shows the wireless networking capability and the mini-server's database system of the platform. The mini-server provides networking services via wired and wireless LAN. Users can use PDAs or PCs to access these services.

The software architecture of this system is given as figure 2. The operating system that we chose to use is Linux. The main advantages of using Linux are its configurability, source code availability, vendor independence, and low cost. All development tools and OS components are available free of charge. Also, many free applications and utilities can be found in open source community.

The software architecture integrates a web server, a mini-database, and some networking components (such as mobile IP, DHCP, DNS Proxy, etc.) Users can use the networking components to access the wireless LAN or the Internet.

All services that need to interact with users are supported by HTML user interfaces. So users can easily access these services using their PDAs or PCs with built-in WiFi communication and web browser. There is no need to install extra customized software in their device.

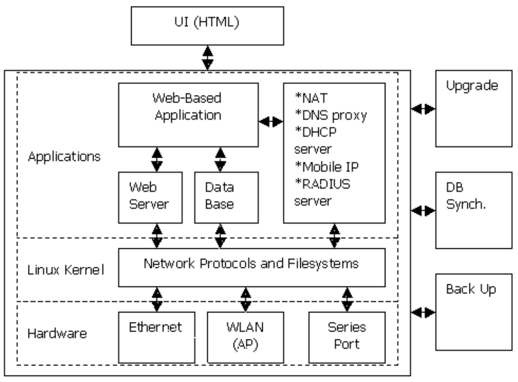


Figure 2.
The software architecture

Our goal is to make the mini-server platform very low in cost and power consumption, and very small in dimension so that it can be fit into a small space. This is very useful in some applications. For example, when using this system in a restaurant, the system can be as small as a wireless access point hanging on the ceiling. Waiters can use the portable PDAs to send orders of customers and stored in the server. Cook in the kitchen can check orders using web interfaces. Customers can access Internet and check their e-mail accounts via the wireless networking support in the restaurant. Countermen can query the database using web interface to get the payment information of each order. In addition, managers get the statistical information of customers' profiles from the integrated database services. All the functions could be served by the platform without the need to install additional servers.

3 Cross-Development Environment

3.1 Host and Target

There are two main development environments for the implementation of the embedded systems: host and s target.

The host is the environment where developers develop their embedded software applications on, for example, a desktop PC with Linux installed. On the other hand, the target is the embedded system itself. In general, developers do development on the host, and the resulted software will be put onto the target and then be executed.

There are several methods to transfer information between the host and the target. The target uses a Compact Flash card as its main storage media, which can also be mounted to the host. During the initial phase to construct the file system and the kernel services of the target, the Compact Flash card is programmed and configured on the host, and is loaded by the target. When the networking services are ready to run on the target, we can use ftp to update the content of the target's file system and use telnet to control and monitor the behavior of the target. In this case, no physical hardware storage device is being transferred between the host and the target.

3.2 uClibc Toolchain

The toolchain to cross-develop applications for the target includes the binary utilities, the C compiler, and the C library [1]. The binary utilities and the C compiler are executables of the host, and they generate executables of the target. The C library is used both on the host and on the target. On the host, it is linked during the compiling time. On the target, it is loaded while the executables are linking to C library dynamically. If all the binaries on the target are linking to C library statically, there is no need to put a copy of C library on the target.

In our development environment, the GNU binary utilities, binutils, and the GNU C compiler, gcc, are adopted. Instead of using the GNU C library (glibc), the uClibc is more applicable C library when considering the size limitation of embedded systems. Functions and function features that are seldom used are omitted from uClibc. Nevertheless, most applications that can be compiled against the glibc will also be able to be compiled and run using uClibc.

On the official web site of uClibc, there is a package that can build the complete uClibc toolchain conveniently. This package contains a set of Makefiles instead of all needed source files. When the user unpacks the package and issues the "make" instruction, it will download the necessary source files from Internet, apply appropriate patches, and build the complete toolchain, including the GNU binary utilities, the GNU C compiler, and uClibc.

By default, the toolchain will be generated to the directory *toolchain_i386*. Under this directory, the most important two sub-directories are *bin* and *lib*.

The directory *toolchain-i386/bin* contains all the utilities within the cross-development toolchain. The names of these utilities are prepended by the "i386-linux-", such as *i386-linux-gcc*, *i386-linux-as*, and *i386-linux-strip*. These utilities are actually executable on the host, and they will generate binaries executable on the target.

The directory *toolchain-i386/lib* contains the runtime libraries for the target. When we compile applications for the target on the host, the compiler will reference the libraries in this directory to generate binaries. If there are some binaries those link to C library dynamically, it is needed to copy this directory to the target's file system.

To use the toolchain to generate binaries of the target, issue some instruction like the following one:

```
(Host) $ i386-linux-gcc –o a.o a.c
```

The binary *a.o* is object code on the target.

3.3 Developing Applications using the Environment

There are five main steps to build applications of the target: download and extract the source package, read installation document, configure, compile, and install and test. The previous four steps are operated on the host, and the last one is operated on the target.

The third step, configure, is the most important step among the five. There are three main configuration methods, and which one should be used depends on the package type.

The first configuration method is to modify the Makefile directly. This is used for the package that has only source files and Makefiles, and the Makefiles are used to compile the source files only. In such case, we first locate the "CC =" line in the Makefile. "CC = gcc" is the most possible situation. Then modify the line to "CC = i386-linux-gcc." By this modification, the package will be compiled with the uClibc toolchain.

The later one is to pass some appropriate options to the configure script. This is used for the package that uses autoconf to check the system environment and generate Makefiles for each tool in the package automatically. For example, we issue the following instruction in some package's directory:

```
(Host) $ CC=i386-linux-gcc \
> ./configure --target=i386-linux
```

Then the configure script will generate the Makefiles that use the *i386-linux-gcc* to compile. Use "./configure --help" to see all options available of the package's configure script.

The last one is to pass some appropriate options to the Makefile. This is used for the package that use the curses-based terminal configuration menu. For example, we issue the following command:

```
(Host) $ make CROSS=i386-linux- \
> menuconfig
```

The configuration system will generate a *.config* file based on the settings we choose. This configuration system is used in Linux kernel.

4 Main System Components

4.1 Linux Kernel

The compilation of linux kernel is a good example to understand the cross-development environment. And it gives us a chance to test the cross-development toolchain built earlier.

The kernel is the central software component of the Linux systems. Linux kernel 2.4.22 is chosen in our project to be used on the target. Because the target belongs to i386 architecture, we can download the kernel source from the main repository, http://www.kernel.org/, directly.

There are four main steps of building a customize Linux kernel:

1. Configure the kernel source
2. Build dependencies and kernel image
3. Build kernel modules
4. Install the kernel image and modules to the target

These steps are much the same as we do when building a new kernel for our workstation. The only difference is that the configuration is chosen for the target platform, and the cross-development toolchain is involved to build the kernel.

Unlike the kernel image that is always loaded in memory, the kernel modules will be loaded into kernel space if necessary. We can set some kernel features to be built as modules if they are needed but are rarely used. This makes the embedded Linux kernel more flexible.

4.2 Bootloader

The bootloader is the first software to be executed when the target system starts up. It is responsible for loading the OS, that is, the Linux kernel we built.

On the target, LILO is chosen as bootloader. To install LILO on the host, first create a LILO configuration file. This file must describe the information about how to install the LILO and how to boot the target. Then use *lilo* command to install LILO on the CF card that will be used on the target.

For more information about the installation of LILO, please reference the online guild in the LILO mini-HOWTO website [5].

If everything boots up normally, the CF card with Linux kernel and bootloader is ready for booting the target system.

4.3 BusyBox and TinyLogin

The BusyBox combines tiny versions of many UNIX utilities into a single small executable [6]. It provides many commands we may use in the system, such as *cat*, *cp*, *kill*, *ps*, *vi*, etc.

The TinyLogin combines many login utilities into a single small executable, and it is often used with BusyBox. Both packages use the same method to provide command-line support. There is only one executable and many symbolic links installed for each package. Such symbolic links are linked to the single executable. When the user issues a command, the executable is invoked via the symbolic link, and the executable determines the actual command.

4.4 C Library

If we compile programs using dynamic linking, it is necessary to copy the C library to the target. These library files were built during the compilation of the uClibc toolchain. All we need to do is to copy the library files in the toolchain to the */lib* directory of the target's root file system, that is, the */lib* directory of the CF card.

5 Major Applications

5.1 Networking Services

Table 1 gives a list of the networking services on the target.

Package	Description
inetd	The internet super-server.
telnetd	Network telnet login server.
ftpd	FTP server.
HostAP	Wireless LAN card and access point driver.
wireless tools	Wireless configuration tools.
bridging utilities	Bridging tools.
uDHCP	DHCP server and client.
IPTables	System NAT table configuration tool.
tHTTPd	HTTP server with CGI support.

Table 1.
Integrated networking services

In Linux system, most networking services are provided as daemons. Each daemon listens on a particular port and responds to requests arriving at the port. To provide these networking services, these daemons must be loaded into memory and kept alive before requests coming in. The internet super-server is a special daemon that listens to the ports of all enabled networking services. When a request comes in from a particular port, the corresponding daemon is started, and the request is passed on to it for service [1]. In this case, only the needed daemons are active.

The telnet server helps users to control and monitor the behavior of the target via network. The ftp server makes it easy to update the content of the target's file system via network, too. These two components are used as development and management tools.

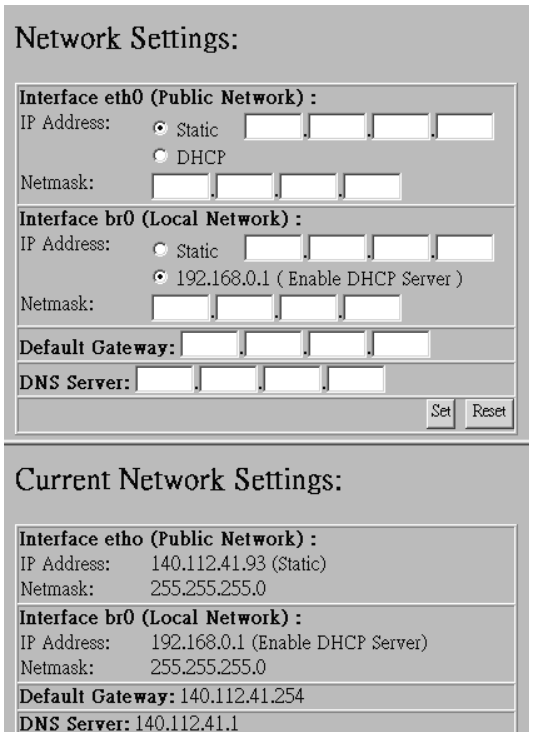
The HostAP and wireless tools provide the capability of wireless service. The target is an access point itself.

The bridging utilities make the wired and the wireless LAN into a single LAN.

The uDHCP provides the services of DHCP server on the LAN. It also provides DHCP client function, and the target can use it to connect to the Internet.

The IPTables provides the interface to configure the system NAT table. Using this tool, the target can act as a NAT and a virtual server.

The tHTTPd is a HTTP server with CGI support. To provide a friendly user interface to users, web pages may be a good choice. The CGI support makes developers can combine other utilities and functions to generate web pages. Figure 3 shows the management CGI to provide networking configuration interface in HTML form. Using the GUI, users do not need to know how to use the *ifconfig* or *route* instruction in command line to configure networking settings. .



Network Settings:

Interface eth0 (Public Network) :
IP Address: [✓ Static] DHCP
Netmask:

Interface br0 (Local Network) :
IP Address: Static [✓ 192.168.0.1 (Enable DHCP Server)]
Netmask:

Default Gateway:

DNS Server:

Current Network Settings:

Interface etho (Public Network) :
IP Address: 140.112.41.93 (Static)
Netmask: 255.255.255.0

Interface br0 (Local Network) :
IP Address: 192.168.0.1 (Enable DHCP Server)
Netmask: 255.255.255.0

Default Gateway: 140.112.41.254

DNS Server: 140.112.41.1

Figure 3.
A CGI example

5.2 Database and CGI Capability

The Berkeley database manager, Berkeley DB, is chosen to provide sufficient database capability. It is actual a library that manages database files with key and data pairs. Compared with GNU database manager, gdbm, Berkeley DB also supports storing, fetch, and deletion operations by key. In addition, Berkeley DB supports secondary indices and duplicate keys. These advance features make us choose the Berkeley DB instead of gdbm. As a result, we can use other attributes in the data filed to search the database.

With database capability, developers can implement some management functions, such as monthly statistical information about business and customers' consuming behaviors on the target.

Furthermore, with the CGI capability, developers can design friendly user interface in HTML format to provide the integrated database functions. Then users can access these functions using their web browser easily.

6 Summary and Future Work

Figure 4 illustrates the networking services configuration of the target system.

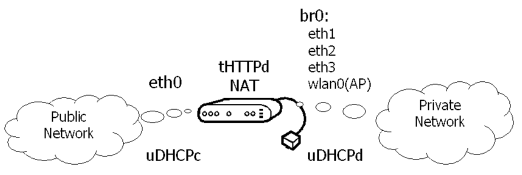


Figure 4.
The networking services configuration

All software packages have been integrated into the embedded mini-server platform. Using the platform, we are designing and implementing a restaurant ordering system.

Furthermore, the system can act as a NAT, a virtual server, a broadband IP router, and an access point for home and small office applications. The necessary software components of these services have been integrated into the platform, and the only thing need to do is to configure these components well.

There are still some issues must be taken into consideration. Because of the limitation of available read/write times of Compact Flash technology, it may be a potential problem if we put all files in the Compact Flash, including the log files and database files. The files which will be modifies frequently should be moved to another storage device, such as an IDE hard drive. Another possible issue is the reliability when the loading of the device is heavy.

The system has been proven to be very easy to use and configurable through web pages. In the future, we will add security mechanism to identify and authenticate users of different roles, namely, customers, waiters, cook, countermen, and managers, etc. We will also add functions to improve the reliability of the system. At the last, all software and hardware components should be reviewed and be stripped away if they were not used. It makes the device simpler and cost lower.

Reference

[1] Ka Karim Yaghmour. *Building Embedded Linux System*. O'Reilly, April 2003.

[2] H. M. Deitel and P. J. Deitel. "Chapter 16: Web Programming with CGI." C++ *How to Program, Fourth Edition*. Prentice Hall, 2003.

[3] The uClibc Website http://www.uclibc.org

[4] The Linux Kernel Website http://www.kernel.org

[5] LILO mini-HOWTO http://tldp.org/HOWTO/LILO.html

[6] The BusyBox Website http://www.busybox.net

[7] The TinyLogin Website http://tinylogin.busybox.net

[8] The HostAP Website http://hostap.epitest.fi

[9] Wireless Tools for Linux http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html

[10] The Ethernet Bridging Utilities http://bridge.sourceforge.net

[11] The uDHCP Website http://udhcp.busybox.net

[12] The tHTTPd Website http://www.acme.com/software/thttpd

[13] The GDBM Online Manual http://cclib.nsu.ru/projects/gnudocs/gnudocs/gdbm/gdbm_toc.html

[14] The Berkeley DB Website http://www.sleepycat.com/products/featurelist.shtml